PUT and POST Requests

Beyond GET: Sending Data to the Server

HTTP Methods Review

Method	Purpose	Data Location	Example Use
GET	Retrieve data	URL parameters	View user profile
POST	Create new data	Request body	Create new user
PUT	Update/replace data	Request body	Update user info
DELETE	Remove data	URL parameters	Delete user

Today's Focus: POST and PUT - sending data in the request body

GET vs POST/PUT: Where Data Goes

- To request a web server, we can choose between GET and POST.
  - GET uses URI.
  - POST uses a hidden request body.

- GET Request Header

```
GET /api/users?name=John&age=25 HTTP/1.1
Host: localhost:8000
```

Data location: URL parameters (visible, limited size)

- POST/PUT Request Header

```
POST /api/users HTTP/1.1
Host: localhost:8000
Content-Type: application/json
Content-Length: 35

{"name": "John", "age": 25}
```

Data location: Request body (hidden, unlimited size)

- We can use other HTTP Methods similarly in the request header.
    - PUT
    - DELETE

Making Requests

- We can make requests in many ways
  - cURL
  - Web Browser
  - JavaScript

cURL

- GET
- POST
- PUT

cURL/GET

- When there is no option, we make a GET request.

```
curl "http://localhost:8000/api/users?name=John%20Doe&email=john@example.com&age=30"
```

cURL/POST

- **Form**: We can use a form to request the same information.

```
curl -X POST http://localhost:8000/api/users \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "name=John Doe&email=john@example.com&age=30"
```

- **Json**: We can request a JSON format.
    - We use JSON in the request body (-d).
    - We use -H to specify the format.

```
curl -X POST http://localhost:8000/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "John Doe", "email": "john@example.com", "age": 30}'
```

- **File Upload**

```
curl -X POST http://localhost:8000/api/upload \
  -F "file=@document.pdf" \
  -F "description=Important document"
```

cURL/PUT

- **JSON**: We can request an update with JSON using the PUT method.

```
curl -X PUT http://localhost:8000/api/users/123 \
  -H "Content-Type: application/json" \
  -d '{"name": "John Smith", "email": "johnsmith@example.com"}'
```

- **Replace Entire Resource**

```
curl -X PUT http://localhost:8000/api/users/123 \
  -H "Content-Type: application/json" \
  -d '{"id": 123, "name": "John Smith", "email": "john@example.com", "age": 31}'
```

PUT vs POST: PUT typically replaces the entire resource, POST creates a new one

Web Browser

- GET
- POST

Browser limitation: HTML forms can only send GET and POST!

HTML Forms (GET)

```
<form action="/api/users" method="GET">
  <input type="text" name="name" placeholder="Full Name">
  <input type="email" name="email" placeholder="Email">
  <button type="submit">Search</button>
</form>
```

HTML Forms (POST)

```
<form action="/api/users" method="POST" enctype="multipart/form-data">
    <input type="text" name="name" placeholder="Full Name">
    <input type="email" name="email" placeholder="Email">
    <input type="file" name="avatar">
    <button type="submit">Create User</button>
</form>
```

JavaScript

- JavaScript can define and use methods for various tasks.
    - One of the most common is the **Fetch API**, which allows JavaScript to make HTTP requests to servers.
- JavaScript is also widely used in **AJAX (Asynchronous JavaScript and XML)** to retrieve data from a server and update parts of a web page **without reloading the entire page**.

Using Fetch API (Modern)

- **POST request**

```
fetch('/api/users', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({
        name: 'John Doe',
        email: 'john@example.com'
    })
});
```

- **PUT request**

```
fetch('/api/users/123', {
    method: 'PUT',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({
        name: 'John Smith',
        email: 'johnsmith@example.com'
    })
});
```

Content-Type for POST

Using the POST method, we can include various content formats in the request body, such as:

- JSON (e.g., application/json)
- Form data (e.g., application/x-www-form-urlencoded)
- Plain text (e.g., text/plain)
- XML (e.g., application/xml)
- Multipart form data for file uploads (e.g., multipart/form-data)

JSON

```
POST /api/users HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0...
Content-Type: application/json
Content-Length: 58
Accept: application/json

{"name": "John Doe", "email": "john@example.com", "age": 30}
```

Key Parts:

- **Headers**: Metadata (Content-Type, Content-Length)
- **Empty Line**: Separates headers from body
- **Body**: The actual data being sent

File Upload

```
POST /api/upload HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="name"

John Doe
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename="document.pdf"
Content-Type: application/pdf

[Binary file data here]
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Form Data

- Only Content-type and contents are shown for the rest of the content types.

```
Content-Type: application/x-www-form-urlencoded

name=John&age=30&skills%5B%5D=PHP&skills%5B%5D=JavaScript
```

Plain Text

```
Content-Type: text/plain

This is plain text data being sent to the server.
```

XML

```
Content-Type: application/xml

<user><name>John</name><age>30</age></user>
```

Handling Various Content Types in the Server

- The PHP should process the requests from users through cURL, web browsers, JavaScript, and many others.
- Then, how is POST (PUT) Data processed in the Request?

Multimedia

- In this example, the request contains binary file information.

```
POST /api/users HTTP/1.1
Host: localhost:8000
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Length: [calculated automatically]

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="name"

John Doe
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="email"

john@example.com
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="avatar"; filename="photo.jpg"
Content-Type: image/jpeg

[binary file content here]
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

```php
// For application/x-www-form-urlencoded and multipart/form-data
$name = $_POST['name'];           // "John"
$email = $_POST['email'];         // "john@example.com"
$uploadedFile = $_FILES['avatar']; // File information

// Check if POST data exists
if (!empty($_POST)) {
    echo "Received POST data: " . print_r($_POST, true);
}
```

JSON

- This is an example that cURL makes a POST request of JSON.

```
curl -X POST http://localhost:8000/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "John Doe", "email": "john@example.com"}'
```

- PHP doesn't populate $_POST for JSON or PUT requests
- You need to read raw input

```php
$raw_data = file_get_contents('php://input');
$json_data = json_decode($raw_data, true);

$name = $json_data['name'];   // "John Doe"
$email = $json_data['email']; // "john@example.com"
```

Why Use `php://input`?

- $_POST only works for (a) - application/x-www-form-urlencoded or (b) - multipart/form-data.
- For everything else, use php://input

```php
$raw_data = file_get_contents('php://input');

// Parse based on Content-Type
$content_type = $_SERVER['CONTENT_TYPE'] ?? '';

if (strpos($content_type, 'application/json') !== false) {
    $data = json_decode($raw_data, true);
} elseif (strpos($content_type, 'application/xml') !== false) {
    $data = simplexml_load_string($raw_data);
} else {
    // Handle other types
    $data = $raw_data;
}
```

Real-World Example

- In most PHP server applications, we typically need to handle two main types of information:
- Request Method Handling (e.g., GET, POST, PUT, DELETE)
- Endpoint Routing (e.g., determining which API path or URL to respond to)

Handling Methods

```php
<?php
$method = $_SERVER['REQUEST_METHOD'];

switch ($method) {
    case 'GET':
        // Handle GET parameters
        $id = $_GET['id'] ?? null;
        echo "Getting user: " . $id;
        break;

    case 'POST':
        // Handle form data
        if (!empty($_POST)) {
            $name = $_POST['name'];
            echo "Creating user: " . $name;
        } else {
            // Handle JSON data
            $json = json_decode(file_get_contents('php://input'), true);
            $name = $json['name'];
            echo "Creating user from JSON: " . $name;
        }
        break;

    case 'PUT':
        // Always read raw input for PUT
        $json = json_decode(file_get_contents('php://input'), true);
        $name = $json['name'];
        echo "Updating user: " . $name;
        break;
}
?>
```

Handling API Endpoint + Method

```php
<?php
header('Content-Type: application/json');

$method = $_SERVER['REQUEST_METHOD'];
$path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);

switch ($path) {
    case '/api/users':
        if ($method === 'GET') {
            // Return list of users
            echo json_encode(['users' => ['John', 'Jane']]);
        } elseif ($method === 'POST') {
            // Create new user
            $input = json_decode(file_get_contents('php://input'), true);
            $name = $input['name'] ?? 'Unknown';
            echo json_encode(['message' => "Created user: $name"]);
        }
        break;

    case '/api/users/123':
        if ($method === 'PUT') {
            // Update user 123
            $input = json_decode(file_get_contents('php://input'), true);
            $name = $input['name'] ?? 'Unknown';
            echo json_encode(['message' => "Updated user 123: $name"]);
        }
        break;

    default:
        http_response_code(404);
        echo json_encode(['error' => 'Not found']);
}
?>
```

PHP File Upload Handling Example

- HTML Form to upload a file.
- As an example, a user selects a file "notes.pdf"
- A browser makes the request, and the `upload.php` handles the request.

```
<form action="upload.php" method="POST" enctype="multipart/form-data">
    <input type="file" name="document">
    <input type="text" name="description">
    <button type="submit">Upload</button>
</form>
```

- The browser automatically generates this header.

```
POST /upload.php HTTP/1.1
Host: yoursite.com
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryabc123
Content-Length: [calculated automatically]

------WebKitFormBoundaryabc123
Content-Disposition: form-data; name="description"

My notes PDF
------WebKitFormBoundaryabc123
Content-Disposition: form-data; name="document"; filename="notes.pdf"
Content-Type: application/pdf

[binary file content here]
------WebKitFormBoundaryabc123--
```

- This is the `upload.php` that processes the request.

```php
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $description = $_POST['description'];

    if (isset($_FILES['document'])) {
        $file = $_FILES['document'];

        echo "File name: " . $file['name'] . "\n";
        echo "File size: " . $file['size'] . " bytes\n";
        echo "File type: " . $file['type'] . "\n";
        echo "Temp path: " . $file['tmp_name'] . "\n";
        echo "Description: " . $description . "\n";

        // Move uploaded file to permanent location
        if (move_uploaded_file($file['tmp_name'], 'uploads/' . $file['name'])) {
            echo "File uploaded successfully!";
        }
    }
}
?>
```

Testing Your API with cURL

```
# Test POST
curl -X POST http://localhost:8000/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Alice", "email": "alice@example.com"}'

# Test PUT
curl -X PUT http://localhost:8000/api/users/123 \
  -H "Content-Type: application/json" \
  -d '{"name": "Alice Smith", "email": "alice.smith@example.com"}'

# Test file upload
curl -X POST http://localhost:8000/api/upload \
  -F "file=@test.txt" \
  -F "description=Test file"
```

Key Differences Summary

Aspect	GET	POST	PUT
Data Location	URL parameters	Request body	Request body
PHP Access	`$_GET`	`$_POST` or `php://input`	`php://input`
Browser Support	✅ Full	✅ Full	❌ Forms only
Typical Use	Read data	Create data	Update data
Idempotent	✅ Yes	❌ No	✅ Yes
Cacheable	✅ Yes	❌ No	❌ No

Key Takeaways

1. **GET**: Data in URL, limited size, visible
2. **POST/PUT**: Data in body, unlimited size, hidden
3. **Browsers**: Limited to GET/POST in forms, need JavaScript for PUT
4. **PHP Handling**:
   - `$_POST` for form data
   - `php://input` for JSON/XML/PUT
   - `$_FILES` for file uploads
5. **Content-Type** determines how to parse the data

Next: Building complete CRUD APIs with PHP and MySQL!